

Graduate Institute of Communication Engineering  
College of Electrical Engineering and Computer Science  
National Taiwan University  
Term Paper

# **An Implementation for JPEG Decoder**

Student Name: Ying-Wun Huang  
Student ID: R00942033

Student Name: Yi-Fan Chang  
Student ID: R00942127

Advisor: Jian-Jiun Ding, Ph.D.

June, 2012

# CONTENTS

CONTENTS .....	i
LIST OF FIGURES .....	ii
LIST OF TABLES.....	iii
<b>Chapter 1    Introduction.....</b>	<b>1</b>
<b>Chapter 2    The JPEG Decoder .....</b>	<b>3</b>
2.1    Read Header.....	3
2.1.1    Read Quantization Table.....	4
2.1.2    Read Frame Header .....	7
2.1.3    Read Huffman Table .....	8
2.2    Cluster for Huffman Table.....	14
2.2.1    Introduction .....	14
2.2.2    Proposed Method .....	15
2.2.3    Jumping Over the Clusters .....	15
2.2.4    Example of Decoding a Codeword .....	16
2.3    Read Bitstream of Huffman coding.....	18
<b>Chapter 3    Experimental Results and Conclusion .....</b>	<b>20</b>
3.1    How to Run This Program.....	20
3.2    Decoded Result.....	24
3.3    Conclusion .....	24
REFERENCE .....	25

# LIST OF FIGURES

Fig. 1.1	Flow chart of JPEG encoder and decoder. (a) Encoder, (b) Decoder.....	1
Fig. 2.1	lena.jpg.....	3
Fig. 2.2	JFIF and quantization table in JPEG file. ....	4
Fig. 2.3	Zigzag-scanning for 8x8 block. ....	5
Fig. 2.4	Frame header and Huffman table in JPEG file. ....	6
Fig. 2.5	4:2:0 .....	8
Fig. 2.6	Single-side growing Huffman tree (SGH-tree). [1] (a) 5 level, (b) 13 level.	14
Fig. 2.7	Change the 13-level SGH-tree into the super tree (S-tree). [1] .....	15
Fig. 2.8	Example for jumping over the clusters. [1] .....	18
Fig. 2.9	Bitstream of Huffman code in JPEG file.....	19
Fig. 3.1	<i>JPEGDecoderDemo.exe</i> in command window. ....	20
Fig. 3.2	Parts of lena.jpg in <i>Parser.txt</i> . ....	21
Fig. 3.3	Parts of lena.jpg in <i>Parser.txt</i> . ....	22
Fig. 3.4	Details of decoding steps in command window. ....	23
Fig. 3.5	The decoded result. (a) lena.jpg, (b) lena.bmp .....	24

# LIST OF TABLES

Table. 2.1 Some markers of the JPEG header .....	4
Table. 2.2 Quantization table for luminance .....	5
Table. 2.3 Quantization table for chrominance. ....	6
Table. 2.4 Number of codeword v.s. codelength for Huffman table for luminance DC...9	
Table. 2.5 Huffman table for luminance DC.....	10
Table. 2.6 Number of codeword v.s. codelength for Huffman table for luminance AC. 10	
Table. 2.7 Part of Huffman table for luminance AC. ....	11
Table. 2.8 Number of codeword v.s. codelength for Huffman table for chrominance DC.12	
Table. 2.9 Huffman table for chrominance DC .....	12
Table. 2.10 Number of codeword v.s. codelength.....	13
Table. 2.11 Part of Huffman table for chrominance AC. ....	13
Table. 2.12 SGH table. [1].....	16

# Chapter 1 Introduction

Generally speaking, the JPEG decoder is the inverse of JPEG encoder as shown in Fig. 1.1.

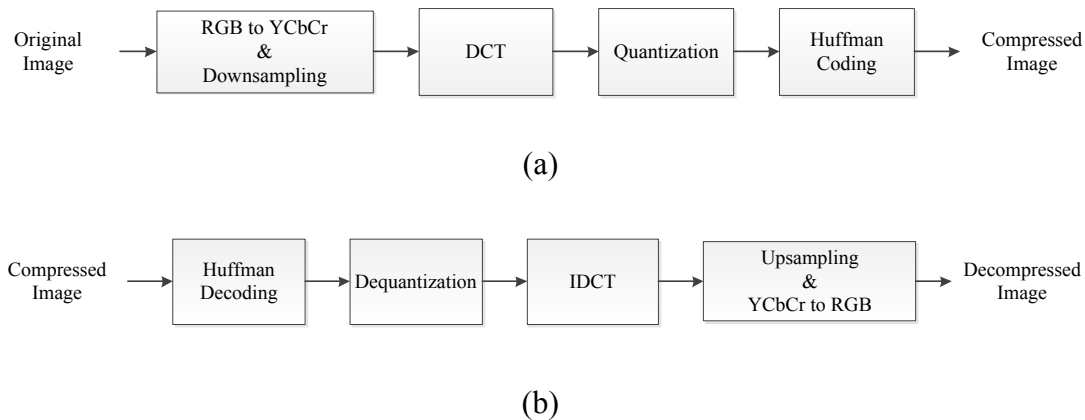


Fig. 1.1 Flow chart of JPEG encoder and decoder. (a) Encoder, (b) Decoder.

A brief description of the JPEG encoder:

1. Reading the pixel value of the original image in RGB color space.
2. Transfer the RGB color space to YCbCr space since the human eyes are sensitive to luminance than chrominance which can be downsampled (ex. 4:2:0) to save the storage.
3. Transfer the YCbCr space from space domain to frequency domain by discrete cosine transform (DCT) since the human eyes are sensitive to low frequency than high frequency which can be hardly quantized to save the storage.
4. Using quantization to decrease the DCT coefficient values required to recode.
5. Encode the bitstream by Huffman coding due to the probability distribution of symbols. Symbols with high probability will have shorter codelength, which is a good property to decrease the memory usage.

A brief description of the JPEG decoder:

1. Reading the bitstream and decode by Huffman decoding.
2. Dequantize the quantized DCT coefficient values.
3. Do the inverse DCT (IDCT) to transfer the YCbCr space from frequency domain to space domain.
4. Reconstruct the chrominance by upsampling and transfer the YCbCr space to RGB color space.
5. Save the pixel value to the bitmap in RGB space and output the decompressed image.

In the following chapters, we will discuss the JPEG decoder in details and show the experimental results.

## Chapter 2 The JPEG Decoder

In this chapter, we will introduce the JPEG decoder step by step. Here we give an example of decoding a JPEG image “lena.jpg” compressed by Paint on Windows 7.



Fig. 2.1 lena.jpg

### 2.1 Read Header

For the purpose of discussing the content of a JPEG file, we show the file in hexadecimal with PSPad which is a text editor (available at <http://www.pspad.com/en/download.php>).

From Fig. 2.2, we can find that it comes with “FF DB” in the beginning, which is the marker “SOI”. The meaning of the marker can be found in Table. 2.1. “FF E0” comes after “FF D8”, which is the beginning of the JFIF header (information about JFIF can be found at [http://en.wikipedia.org/wiki/JPEG\\_File\\_Interchange\\_Format](http://en.wikipedia.org/wiki/JPEG_File_Interchange_Format)).

After the JFIF header, we will start to recode 6 important tables which are quantization table for luminance, quantization table for chrominance, Huffman table for luminance DC, Huffman table for luminance AC, Huffman table for chrominance DC and Huffman table for chrominance AC.

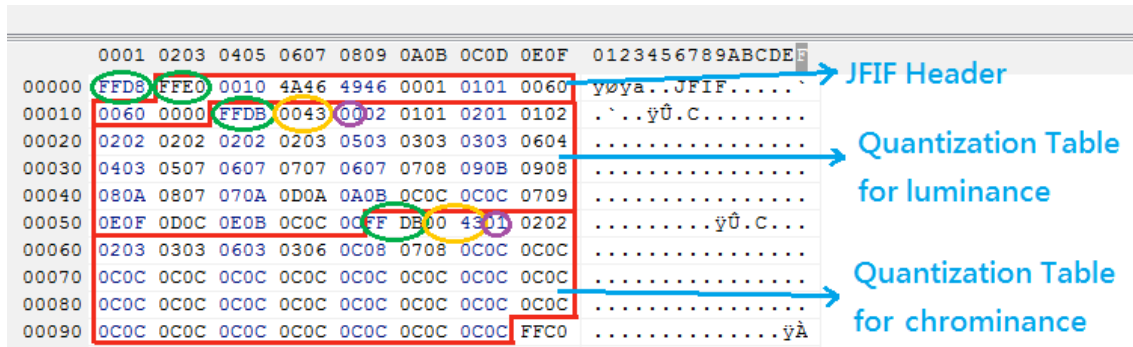


Fig. 2.2 JFIF and quantization table in JPEG file.

Table. 2.1 Some markers of the JPEG header

Name	Code (HEX)	Description
SOI	FFD8	Start Of Image
EOI	FFD9	End Of Image
SOF	FFC0	Start Of Frame
SOS	FFDA	Start Of Scan
DHT	FFC4	Define Huffman Table
DQT	FFDB	Define Quantization Table
	FFE0	JFIF marker

### 2.1.1 Read Quantization Table

From Fig. 2.2 and Table. 2.1, the quantization tables are start from “FF DB”.

For the quantization table – luminance:

- “00 43”: The length of quantization table is  $43 - 2(\text{“00 43”}) - 1(\text{“00”}) = 40(\text{Hex}) = 64(\text{Dec})$ .
- “00”: The marker of quantization table for luminance.
- “02 01”~”0C 0C”: The content of quantization table for luminance, which



should be record in zigzag form (Fig. 2.3) as shown in Table. 2.2.

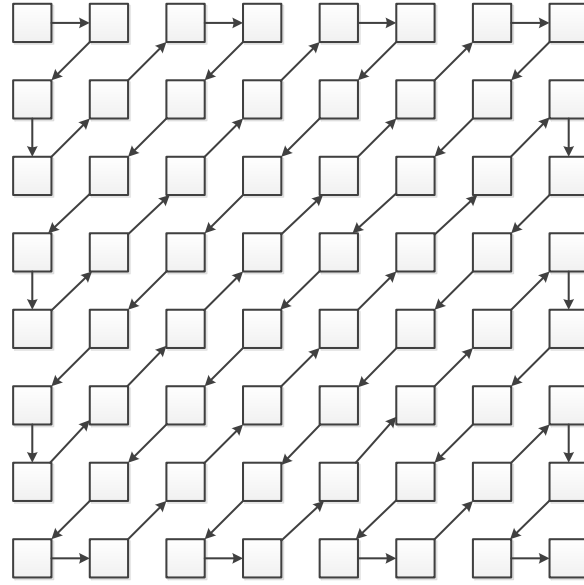


Fig. 2.3 Zigzag-scanning for 8x8 block.

Table. 2.2 Quantization table for luminance

2	1	1	2	3	5	6	7
1	1	2	2	3	7	7	7
2	2	2	3	5	7	8	7
2	2	3	3	6	10	10	7
2	3	4	7	8	13	12	9
3	4	7	8	10	12	14	11
6	8	9	10	12	15	14	12
9	11	11	12	13	12	12	12

For the quantization table –chrominance:

- “00 43”: The length of quantization table is  $43-2(\text{“00 43”})-1(\text{“00”})=40(\text{Hex})$   
=64(Dec).
- “01”: The marker of quantization table for chrominance.
- “02 02”~”0C 0C”: The content of quantization table for chrominance, which



### 2.1.2 Read Frame Header

“FF C0” means start of the frame header which contains the information about the frame such as frame height, frame width, subsampling mode, etc. The following are the details of frame header in Fig. 2.4:

- “00 11”: The length of the frame header is 11(Hex) = 17(Dec).
- “08”: Range of pixel value is 0~255(8 bit).
- “02 00”: The height of the frame is 200(Hex) = 512(Dec).
- “02 00”: The width of the frame is 200(Hex) = 512(Dec).
- “03”: Number of components is 3 (Y, Cb, Cr).
- “01”: ID of Y.
- “22”: The horizontal sampling factor of Y is  $H_Y=2$ , and the vertical sampling factor of Y is  $V_Y=2$ .
- “00”: Quantization table number of Y.
- “02”: ID of Cb.
- “11”: The horizontal sampling factor of Cb is  $H_{Cb}=1$ , and the vertical sampling factor of Cb is  $V_{Cb}=1$ .
- “01”: Quantization table number of Cb.
- “03”: ID of Cr.
- “11”: The horizontal sampling factor of Cr is  $H_{Cr}=1$ , and the vertical sampling factor of Cr is  $V_{Cr}=1$ .
- “01”: Quantization table number of Cr.

The frame sizes of each component can be calculated from frame height, frame width, and sampling factors. In this case, the maximum horizontal sampling factor is  $H_{max}=2$  and maximum vertical sampling factor is  $V_{max}=2$ .

Thus, the frame sizes of each component are:

- Frame width of Y:  $\text{frame width} \times \frac{H_Y}{H_{\max}} = 512 \times \frac{2}{2} = 512$
- Frame height of Y:  $\text{frame height} \times \frac{V_Y}{V_{\max}} = 512 \times \frac{2}{2} = 512$
- Frame width of Cb:  $\text{frame width} \times \frac{H_{Cb}}{H_{\max}} = 512 \times \frac{1}{2} = 256$
- Frame height of Cb:  $\text{frame height} \times \frac{V_{Cb}}{V_{\max}} = 512 \times \frac{1}{2} = 256$
- Frame width of Cr:  $\text{frame width} \times \frac{H_{Cr}}{H_{\max}} = 512 \times \frac{1}{2} = 256$
- Frame height of Cr:  $\text{frame height} \times \frac{V_{Cr}}{V_{\max}} = 512 \times \frac{1}{2} = 256$

From the sampling factors, it can be found that Cb and Cr are downsampled in every 2 pixels in both vertical and horizontal direction. Therefore, the subsampling mode in this case is 4:2:0 as shown in Fig. 2.5.



Fig. 2.5 4:2:0

### 2.1.3 Read Huffman Table

In this section, we will read the Huffman table in the JPEG file first, and then cluster the Huffman table by the method 0 in section 2.2.

The following are the details of Huffman table for luminance DC in Fig. 2.4:

- “FF C4”: Define Huffman table which is the start marker.
- “00 1F”: The number of category is  $1F - 2(“00 1F”) - 1(“00”) - 10(“00 01\sim 00 00”) = D(\text{Hex}) = 12(\text{Dec})$ .
- “00”: Huffman Table for luminance DC.
- “00 01”~”00 00”: Numbers of codeword with codelength equal to 1~16. The detail of this case is shown in Table. 2.4.
- “00 01”~”0A 0B”: Category of Huffman table for luminance DC.

Now we can generate the Huffman table for luminance DC with Table. 2.4 and algorithm 1. The result is shown in Table. 2.5.

Table. 2.4 Number of codeword v.s. codelength for Huffman table for luminance DC.

Codelength	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of Codeword	0	1	5	1	1	1	1	1	1	0	0	0	0	0	0	0

Algorithm 1: Generate Huffman table
<pre> i = 0, codevalue = 0; for(k = 1; k &lt;= 16; k++) {     for(j = 1; j &lt;= number_of_codeword[k]; j++)     {         codeword[i] = codevalue;         codelength[i] = k;         codevalue++;         i++;     }     codevalue *= 2; } </pre>

Table. 2.5 Huffman table for luminance DC.

Category	Codelength	Codeword
00	2	00
01	3	10
02	3	11
03	3	100
04	3	101
05	3	110
06	4	1110
07	5	11110
08	6	111110
09	7	1111110
0A	8	11111110
0B	9	111111110

The following are the details of Huffman table for luminance AC in Fig. 2.4:

- “FF C4”: Define Huffman table which is the start marker.
- “00 B5”: The number of category is  $B5 - 2(“00 B5”) - 1(“00”) - 10(“00 02 \sim 00 7D”) = A2(\text{Hex}) = 162(\text{Dec})$ .
- “10”: Huffman Table for luminance AC.
- “00 02”~“00 7D”: Numbers of codeword with codelength equal to 1~16. The detail of this case is shown in Table. 2.6.
- “01 02”~“F9 FA”: Category of Huffman table for luminance AC.

Now we can generate the Huffman table for luminance AC with Table. 2.6 and algorithm 1. The result is shown in Table. 2.7.

Table. 2.6 Number of codeword v.s. codelength for Huffman table for luminance AC.

Codelength	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1										1					

Number of codeword	0	2	1	3	3	2	4	3	5	5	4	4	0	0	1	125
-----------------------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

Table. 2.7 Part of Huffman table for luminance AC.

Category	Codelength	Codeword
01	2	00
02	2	01
03	3	100
00	4	1010
04	4	1011
11	4	1100
05	5	11010
12	5	11011
21	5	11100
31	6	111010
41	6	111011
...	...	...
EA	16	1111111111110100
F1	16	1111111111110101
F2	16	1111111111110110
F3	16	1111111111110111
F4	16	111111111111000
F5	16	111111111111001
F6	16	111111111111010
F7	16	111111111111011
F8	16	111111111111100
F9	16	111111111111101
FA	16	111111111111110

The following are the details of Huffman table for chrominance DC in Fig. 2.4:

- “FF C4”: Define Huffman table which is the start marker.
- “00 1F”: The number of category is  $1F - 2(“00 1F”) - 1(“00”) - 10(“00 01 \sim 00 00”) = D(\text{Hex}) = 12(\text{Dec})$ .
- “01”: Huffman Table for chrominance DC.
- “00 03”~”00 00”: Numbers of codeword with codelength equal to 1~16. The

detail of this case is shown in Table. 2.8.

- “00 01”~”0A 0B”: Category of Huffman table for chrominance DC.

Now we can generate the Huffman table for chrominance DC with Table. 2.8 and algorithm 1. The result is shown in Table. 2.9.

Table. 2.8 Number of codeword v.s. codelength for Huffman table for chrominance DC.

Codelength	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of codeword	0	3	1	1	1	1	1	1	1	1	1	0	0	0	0	0

Table. 2.9 Huffman table for chrominance DC

Category	Codelength	Codeword
00	2	00
01	2	01
02	2	10
03	3	110
04	4	1110
05	5	11110
06	6	111110
07	7	1111110
08	8	11111110
09	9	111111110
0A	10	1111111110
0B	11	11111111110

The following are the details of Huffman table for chrominance AC in Fig. 2.4:

- “FF C4”: Define Huffman table which is the start marker.
- “00 B5”: The number of category is  $B5 - 2(“00 B5”) - 1(“00”) - 10(“00 02 \sim 00 7D”) = A2(\text{Hex}) = 162(\text{Dec})$ .
- “11”: Huffman Table for chrominance AC.



- “00 02”~”02 77”: Numbers of codeword with codelength equal to 1~16. The detail of this case is shown in Table. 2.10.
- “00 01”~”F9 FA”: Category of Huffman table for chrominance AC.

Now we can generate the Huffman table for chrominance AC with Table. 2.10 and algorithm 1. The result is shown in Table. 2.11.

Table. 2.10 Number of codeword v.s. codelength  
for Huffman table for chrominance AC.

Codelength	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of codeword	0	2	1	2	4	4	3	4	7	5	4	4	0	1	2	119

Table. 2.11 Part of Huffman table for chrominance AC.

Category	Codelength	Codeword
00	2	00
01	2	01
02	3	100
03	4	1010
11	4	1011
04	5	11000
05	5	11001
21	5	11010
31	5	11011
06	6	111000
12	6	111001
...	...	...
E9	16	1111111111110100
EA	16	1111111111110101
F2	16	1111111111110110
F3	16	1111111111110111
F4	16	1111111111111000
F5	16	1111111111111001

F6	16	1111111111111010
F7	16	1111111111111011
F8	16	1111111111111100
F9	16	1111111111111101
FA	16	1111111111111110

## 2.2 Cluster for Huffman Table

In this section, we will cluster the Huffman table by method 0 for the purpose of memory efficiency and high-speed search Huffman coding.

### 2.2.1 Introduction

For the traditional Huffman coding algorithm,  $2^k$  words in the memory for  $k$ -bit Huffman code wastes lots of memory and takes a long time when decoding (Fig. 2.6(a)), which will be terrible when  $k$  is 13 or higher (Fig. 2.6(b)).

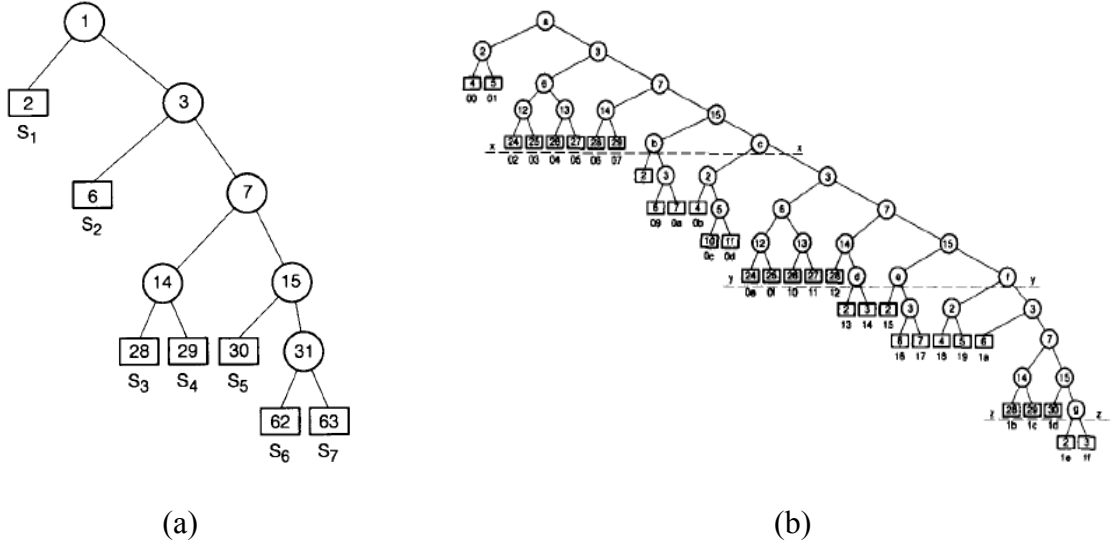


Fig. 2.6 Single-side growing Huffman tree (SGH-tree). 0 (a) 5 level, (b) 13 level.

This paper proposed a tree clustering algorithm based on an ordering and clustering to speed up for searching a symbol in a Huffman tree and reduce the memory

usage. We will discuss the details in the next section.

### 2.2.2 Proposed Method

For the codeword with short length, it costs less searching time due to high probability symbol, but how about the codeword with long length? This paper proposes a method to speed up the searching time by jumping over the clusters (Fig. 2.7).

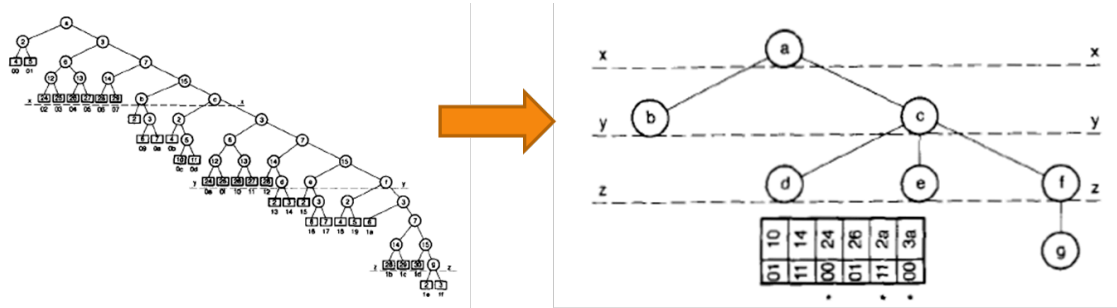


Fig. 2.7 Change the 13-level SGH-tree into the super tree (S-tree). 0

### 2.2.3 Jumping Over the Clusters

Suppose we have a Huffman table as shown in Table. 2.12, and jumping over the clusters by the codelength of 4. Therefore, the Huffman table will be divided into 4 sections by dashed line x, y and z.

In section 1, we imagine that there are four imaginary bits in front of the Huffman codes and each of the four imaginary bits are the same. Thus, the Huffman codes “00” to “1101” are grouped into the same cluster “a”.

In section 2, since there are two kinds of the first four bits “1110” and “1111”, it can be divided into two clusters “b” and “c”. The Huffman codes “11100” to “111011” are grouped into cluster “b”, and “111100” to “11111100” are grouped into cluster “c”.

In section 3, since there are three kinds of the second four bits “1101”, “1110” and “1111”, it can be divided into three clusters “d”, “e” and “f”. For the same reason, we can group the Huffman codes into cluster “g” in section 4.

Table. 2.12 SGH table. 0

CL	Symbols	Huffman Code	
2	00	00	Section 1
2	01	01	
4	02	1000	Section 1
4	03	1001	
4	04	1010	
4	05	1011	
4	06	1100	Section 1
4	07	1101	
5	08	1110	Section 1
6	09	1111	
6	0a	1110	Section 2
6	0b	1111	
6	0c	1111	Section 2
7	0d	1111	
8	0e	1111	Section 2
8	0f	1111	
8	10	1111	Section 2
8	11	1111	
8	12	1111	Section 2
9	13	1111	
9	14	1111	Section 2
9	15	1111	
10	16	1111	Section 3
10	17	1111	
10	18	1111	Section 3
10	19	1111	
12	1b	1111	Section 3
12	1c	1111	
12	1d	1111	Section 3
13	1e	1111	
13	1f	1111	Section 4
13	1f	1111	

## 2.2.4 Example of Decoding a Codeword

Here we give an example to show how to decode a codeword by jumping over the clusters. Assume there is a bitstream “111111111110...” need to decode, and following are the decoding steps as shown in Fig. 2.8:

- Since the cluster length of cluster “a” is  $11+1=100(\text{Bin})=4(\text{Dec})$ , thus we read the first four bits “1111” into the buffer, and “1111(Bin)=15(Dec)” means the 15<sup>th</sup> position which is 12 in the cluster table (a).
- 1 means that we need to look up the super table to see what cluster we should jump to.
- 2 means the 2<sup>nd</sup> position in super table which is 1114.

- $\boxed{11}$  means the cluster length of cluster “c” is  $11+1=100(\text{Bin})=4(\text{Dec})$ .
- $\boxed{14}$  is the start position of cluster “c” in the memory.
- Since the cluster length of cluster “c” is 4, thus we read the next four bits “1111” into the buffer, and “1111(Bin)=15(Dec)” means the 15<sup>th</sup> position which is  $\boxed{1}\boxed{5}$  in the cluster table (c).
- $\boxed{1}$  means that we need to look up the super table to see what cluster we should jump to.
- $\boxed{5}$  means the 5<sup>th</sup> position in super table which is  $\boxed{11}\boxed{2a}$ .
- $\boxed{11}$  means the cluster length of cluster “f” is  $11+1=100(\text{Bin})=4(\text{Dec})$ .
- $\boxed{2a}$  is the start position of cluster “f” in the memory.
- Since the cluster length of cluster “f” is 4, thus we read the next four bits “1111” into the buffer, and “1111(Bin)=15(Dec)” means the 15<sup>th</sup> position which is  $\boxed{1}\boxed{6}$  in the cluster table (f).
- $\boxed{1}$  means that we need to look up the super table to see what cluster we should jump to.
- $\boxed{6}$  means the 6<sup>th</sup> position in super table which is  $\boxed{00}\boxed{3a}$ .
- $\boxed{00}$  means the cluster length of cluster “g” is  $00+1=01(\text{Bin})=1(\text{Dec})$ .
- $\boxed{3a}$  is the start position of cluster “g” in the memory.
- Since the cluster length of cluster “g” is 1, thus we read the one bit “0” into the buffer, and “0(Bin)=0(Dec)” means the 0<sup>th</sup> position which is  $\boxed{0}\boxed{2}$  in the cluster table (g).
- $\boxed{0}$  means that the codeword is in the current cluster.
- $\boxed{2}$  is “10(Bin)”, so the most significant one bit (MS1B) is at the location 1, that is, the codelength is one. The right of MS1B is “0” which means the location

is  $3a+0=3a(\text{Hex})$  in the memory.

- Therefore, the overall codeword is “111111111110”, and the decoded symbol is “1e”.

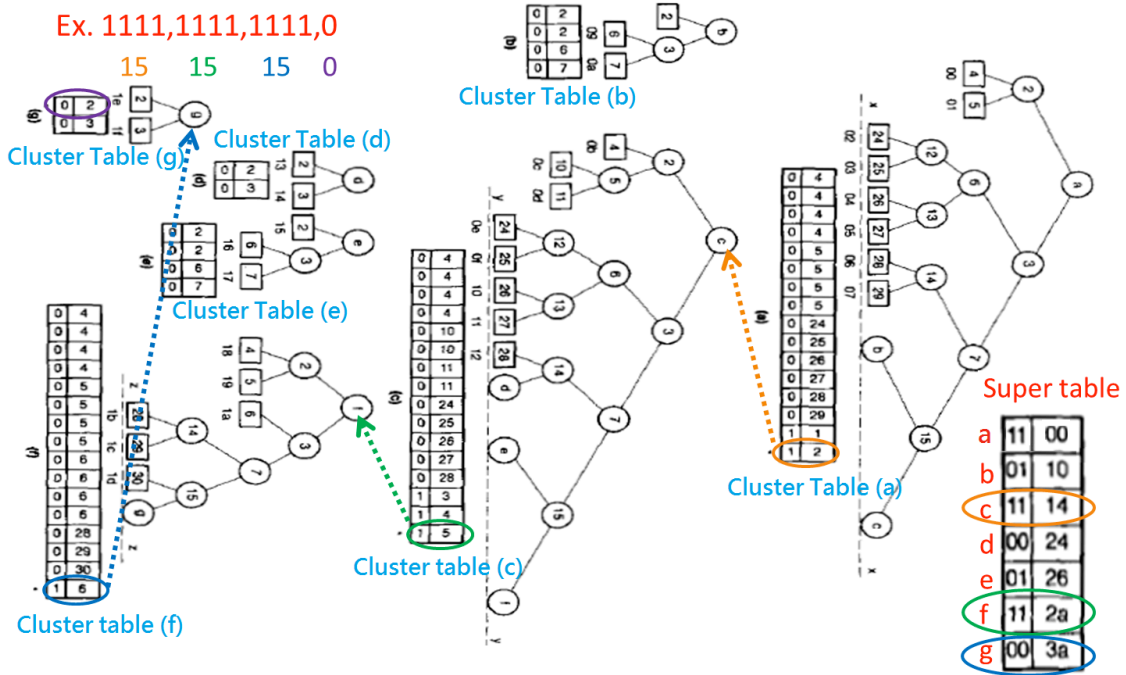


Fig. 2.8 Example for jumping over the clusters. 0

## 2.3 Read Bitstream of Huffman coding

“FF DA” means start of scan which is the start marker of the bitstream of Huffman code, and FF D9“ is the end marker of the bitstream, which means “end of image”. The remains between “FF DA” and “FF D9” are the Huffman codes that we need to decode.

	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
00250	E3E4	E5E6	E7E8	E9EA	F2F3	F4F5	F6F7	F8F9	ãäåæçèéêëðóôõö÷øù
00260	F2FF	DA00	0C03	0100	0211	0311	003F	00FA	úÿÚ.....?.ú
00270	0A5B	840F	D410	BC93	F811	506A	578A	8EC4	. [ „. Œ. ¼”ø. PjWŠŽĂ
00280	8203	A301	D3D8	D569	5196	2DAA	DB48	E73E	, . £. Ó Ø Œ i Q -- * Ũ H ç >
00290	DF4A	2693	7B1E	B82C	724F	F31F	9D7E	353C	ß J & “ { . , , r O ó . ~ 5 <
002A0	4B67	F604	68AD	131B	7374	A238	F83F	2673	K g ö . h - . . s t c 8 ø ? & s
002B0	F8F3	FD2A	1671	E781	91C9	02A5	900D	B825	ø ó ý * . q ç ‘ È . ¥ . , %
002C0	88C7	A67E	B8FE	750C	4864	3C60	B139	0DD4	^ Ç   ~ , p u . H d < ` ± 9 . Ô
⋮									
177D0	7903	AE3A	77AB	3E06	D267	D4EF	2236	E840	y . @ : w « > . Ò g Ô i " 6 è @
177E0	460C	18BF	5EE0	F5EB	FE7A	57BA	FC3E	F857	F . . ¿ ^ à ö ë p z W ° ú > ø W
177F0	169B	A5A6	6343	3AB6	46E0	0FCD	DFDB	9CD5	. > ¥ ! c C : ¶ F à . í ß Ũ æ ð
17800	6331	4B0F	1B9E	9603	0AF1	1514	01FF	D9	c 1 K . . ž - . . ñ . . . Ÿ Ũ

→ Bitstream of Huffman code

Fig. 2.9 Bitstream of Huffman code in JPEG file.

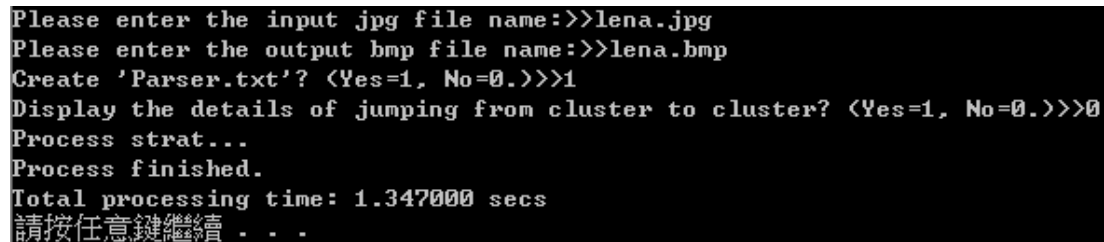
## Chapter 3 Experimental Results and Conclusion

In this chapter, we will show our experimental results. The following are the demo steps of our program, compiled on Windows 7 64bit with Visual Studio 2010.

*JPEGDecoderDemo.exe* is under the folder: *JPEGDecoderDemo\JPEGDecoder Demo\Release\*.

### 3.1 How to Run This Program

Be sure that “*JPEGDecoderDemo.exe*” and JPEG file in the same folder, then double click “*JPEGDecoderDemo.exe*” and enter the input JPEG filename and output bmp filename.



```
Please enter the input jpg file name:>>lena.jpg
Please enter the output bmp file name:>>lena.bmp
Create 'Parser.txt'? <Yes=1, No=0.>>1
Display the details of jumping from cluster to cluster? <Yes=1, No=0.>>0
Process strat...
Process finished.
Total processing time: 1.347000 secs
請按任意鍵繼續 . . .
```

Fig. 3.1 *JPEGDecoderDemo.exe* in command window.

If choosing “Yes” for the question “*Creat 'Parser.txt'?*”, it will create “*Parser.txt*” in the same folder, which shows the information such as quantization tables, Huffman tables, and decoded blocks YCbCr as shown in Fig. 3.2 and Fig. 3.3.



```

***** Quantization Table *****
quantization_table_luminance
  8   6   5   8   12   20   26   31
  6   6   7   10  13   29   30   28
  7   7   8   12   20   29   35   28
  7   9   11  15   26   44   40   31
  9   11  19  28   34   55   52   39
 12   18  28  32   41   52   57   46
 25   32  39  44   52   61   60   51
 36   46  48  49   56   50   52   50

quantization_table_chrominance
  9   9   12   24   50   50   50   50
  9   11  13   33   50   50   50   50
 12   13  28   50   50   50   50   50
 24   33  50   50   50   50   50   50
 50   50  50   50   50   50   50   50
 50   50  50   50   50   50   50   50
 50   50  50   50   50   50   50   50
 50   50  50   50   50   50   50   50

***** Huffman Table *****
-----Huffman Table luminance DC-----
Category  Codelength  Codeword
00         2         00
01         3         010
02         3         011
03         3         100
04         3         101
05         3         110
06         4         1110
07         5         11110
08         6         111110
09         7         1111110
0A         8         11111110
0B         9         111111110

-----Huffman Table luminance AC-----
Category  Codelength  Codeword
01         2         00
02         2         01
03         3         100
00         4         1010
04         4         1011
11         4         1100
05         5         11010
12         5         11011
21         5         11100

```

Fig. 3.2 Parts of lena.jpg in *Parser.txt*.

```

-----Decode Y-----
Before DeQuantize
-20,  -2,  -1,   1,   0,   1,   0,  -1,
-7,   1,   0,   1,   1,   0,   0,   1,
-4,   1,  -1,  -1,   1,   0,   0,   0,
1,    0,   1,   0,   0,   0,   0,   0,
1,   -1,   0,   0,   0,   0,   0,   0,
-1,  -1,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,

Before IDCT
-160, -12,  -5,   8,   0,  20,   0, -31,
-42,   6,   0,  10,  13,   0,   0,  28,
-28,   7,  -8, -12,  20,   0,   0,   0,
7,    0,  11,   0,   0,   0,   0,   0,
9,   -11,   0,   0,   0,   0,   0,   0,
-12, -18,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,

After IDCT
-26, -44, -37, -25, -25, -35, -29, -26,
-16, -30, -28, -20, -25, -32, -32, -36,
-24, -30, -29, -20, -25, -19, -22, -32,
-25, -25, -28, -18, -23, -3,  -6, -18,
-10, -8, -18, -9, -24, 2,  -3, -13,
-13, -9, -21, -7, -30, -3, -11, -14,
-27, -20, -28, -4, -32, -8, -17, -13,
-25, -16, -22, 7, -27, -8, -22, -15,

4 Y blocks are decoded. now
-----Decode Cb-----
Before DeQuantize
-12,   1,   1,   0,   0,   0,   0,   0,
2,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,
0,    0,   0,   0,   0,   0,   0,   0,

Before IDCT
-108, 9,  12,   0,   0,   0,   0,   0,
18,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   0,   0,   0,   0,

```

Fig. 3.3 Parts of lena.jpg in *Parser.txt*.

If choosing “Yes” for the question “*Display the details of jumping from cluster to cluster?*”, then it will display the details of decoding (bitstream in buffer, cluster, decoded symbols, decoded DC terms and AC terms, etc.) step by step in command window as show in Fig. 3.4.

```

-----Reading bitstream -----
1100:received bits in the buffer
buffer = 12
Now we are in the Cluster <a>
current_content=28
The content of the table at this location is 0/28, as the sign/magnitude.
Decoded Symbol: Run/Size 1/1

ACterm = 1
請按任意鍵繼續 . . .

-----Reading bitstream -----
1100:received bits in the buffer
buffer = 12
Now we are in the Cluster <a>
current_content=28
The content of the table at this location is 0/28, as the sign/magnitude.
Decoded Symbol: Run/Size 1/1

ACterm = 1
請按任意鍵繼續 . . .

-----Reading bitstream -----
1110:received bits in the buffer
buffer = 14
Now we are in the Cluster <a>
current_content=-2
The content of the table at this location is 1/2, as the sign/magnitude.
The symbol is not in cluster <a> and refer to location 2 in the ST,
assigned to cluster <c>.

00:received bits in the buffer
buffer = 0
Now we are in the Cluster <c>
current_content=2
The content of the table at this location is 0/2, as the sign/magnitude.
Decoded Symbol: Run/Size 2/1

ACterm = -1
請按任意鍵繼續 . . .

```

Fig. 3.4 Details of decoding steps in command window.

## 3.2 Decoded Result

The following are the compressed image (lena.jpg) and the decompressed image (lena.bmp).



(a)



(b)

Fig. 3.5 The decoded result. (a) lena.jpg, (b) lena.bmp

## 3.3 Conclusion

We have implemented a JPEG decoder with the Huffman decoding method proposed by 0 which claims high efficiency in memory space and high-speed searching for Huffman decoding. Our impression is that it is difficult to deal with the bitstream in the buffer.

## REFERENCE

- [1] R. Hashemian, "Memory Efficient and High Speed Search Huffman Coding", *IEEE Transactions on Communications*, Vol. 43, No. 10, pp. 2576-2581, October, 1995.
- [2] JPEG Standard, "Information Technology- Digital Compression and Coding of Continuous-Tone Still Images Requirements and Guidelines," Recommendation T.81, ITU, September, 1992
- [3] 酒井善則、吉田俊之 共著，白執善 編譯，“影像壓縮技術”，全華，2004.